# Python Basics

April 4th, 2023

# 🐍 Basics

# ~~Python Basics~~

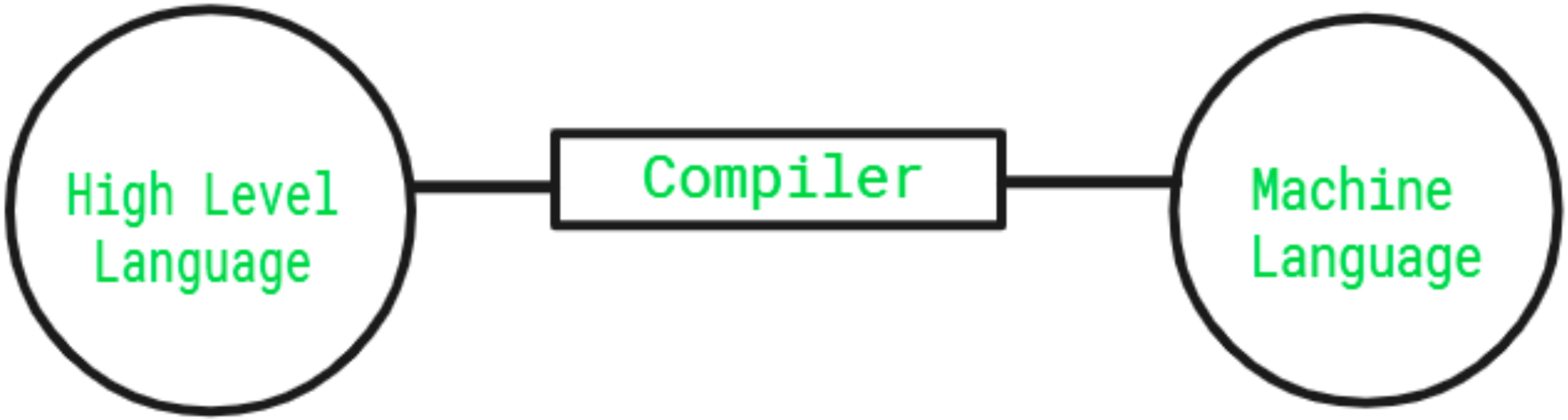
April 4th, 2023

# Today’s Agenda

- Compiled vs. Interpreted
- Variables & Types
- Numbers & Booleans
- Strings
- Lists
- Control Flow

# Compilers

# Compilers

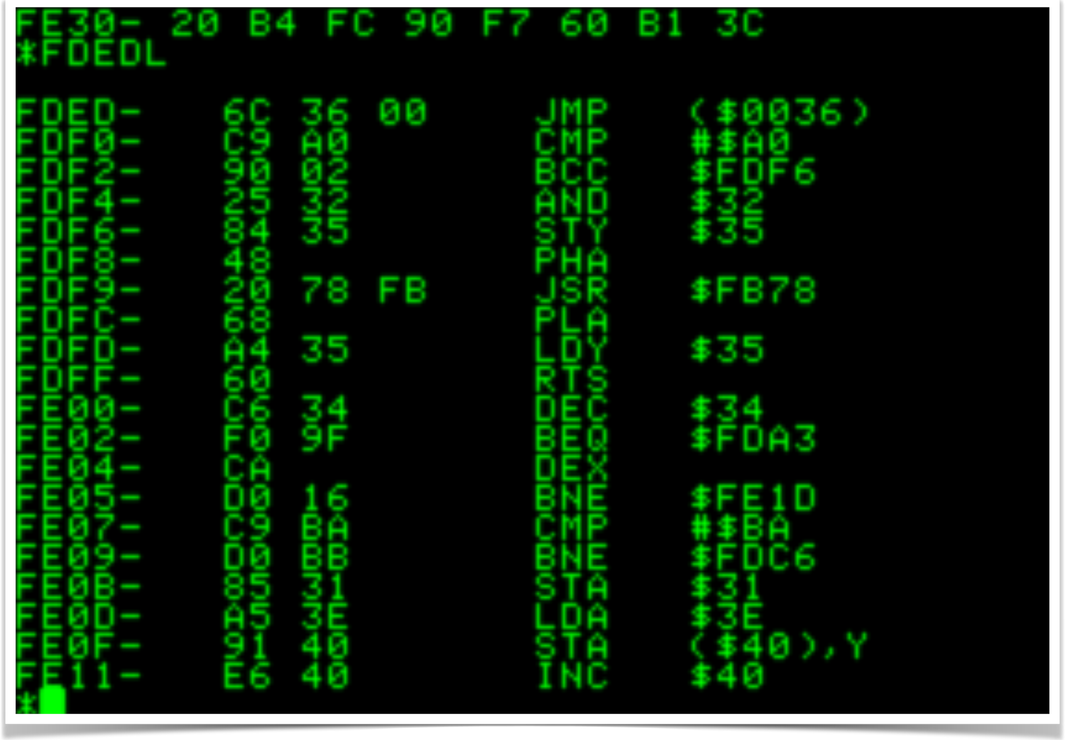



Turns your file into an executable

# What is this “machine code”?

- This is code that directly runs on the CPU, the hardware of the computer

- Hard to read, so that’s why we have programming languages!!



# Some languages are compiled, and some are interpreted

# Some languages are compiled, and some are interpreted

- Instead of sending a CPU an executable file, a program called an interpreter reads your source code line by line, and turns it into machine code then.

# Some languages are compiled, and some are interpreted

- Instead of sending a CPU an executable file, a program called an interpreter reads your source code line by line, and turns it into machine code then.
- This takes out the process of turning your source code into a separate file

# Pros and Cons

## Compiled

- PROS:

# Pros and Cons

## Compiled

- PROS:
  - Once ready to run, it is READY TO RUN!  Can be sent to 1000s of people in its executable state

# Pros and Cons

## Compiled

- PROS:
  - Once ready to run, it is READY TO RUN!  Can be sent to 1000s of people in its executable state
  - Allows others to run your code without source code

# Pros and Cons

## Compiled

- PROS:
  - Once ready to run, it is READY TO RUN!  Can be sent to 1000s of people in its executable state
  - Allows others to run your code without source code
  - Usually faster

# Pros and Cons

## Compiled

- PROS:
  - Once ready to run, it is READY TO RUN!  Can be sent to 1000s of people in its executable state
  - Allows others to run your code without source code
  - Usually faster
- CONS:

# Pros and Cons

## Compiled

- PROS:
  - Once ready to run, it is READY TO RUN!  Can be sent to 1000s of people in its executable state
  - Allows others to run your code without source code
  - Usually faster
- CONS:
  - Executable CPU specific (different for Mac, PC, ect)

# Pros and Cons

## Compiled

- PROS:
  - Once ready to run, it is READY TO RUN!  Can be sent to 1000s of people in its executable state
  - Allows others to run your code without source code
  - Usually faster
- CONS:
  - Executable CPU specific (different for Mac, PC, ect)
  - Extra step to run your code, can make  debugging longer

# Pros and Cons

## Interpreted

- PROS:

# Pros and Cons

## Interpreted

- PROS:
  - Super flexible for cross-platform (every device must interpret code on its own, never sending machine code)

# Pros and Cons

## Interpreted

- PROS:
  - Super flexible for cross-platform (every device must interpret code on its own, never sending machine code)
  - Faster debugging, no extra step

# Pros and Cons

## Interpreted

- PROS:
  - Super flexible for cross-platform (every device must interpret code on its own, never sending machine code)
  - Faster debugging, no extra step
- CONS:

# Pros and Cons

## Interpreted

- PROS:
  - Super flexible for cross-platform (every device must interpret code on its own, never sending machine code)
  - Faster debugging, no extra step
- CONS:
  - Machines must download interpreter themselves

# Pros and Cons

## Interpreted

- PROS:
  - Super flexible for cross-platform (every device must interpret code on its own, never sending machine code)
  - Faster debugging, no extra step
- CONS:
  - Machines must download interpreter themselves
  - Often slower due to need to always interoperate

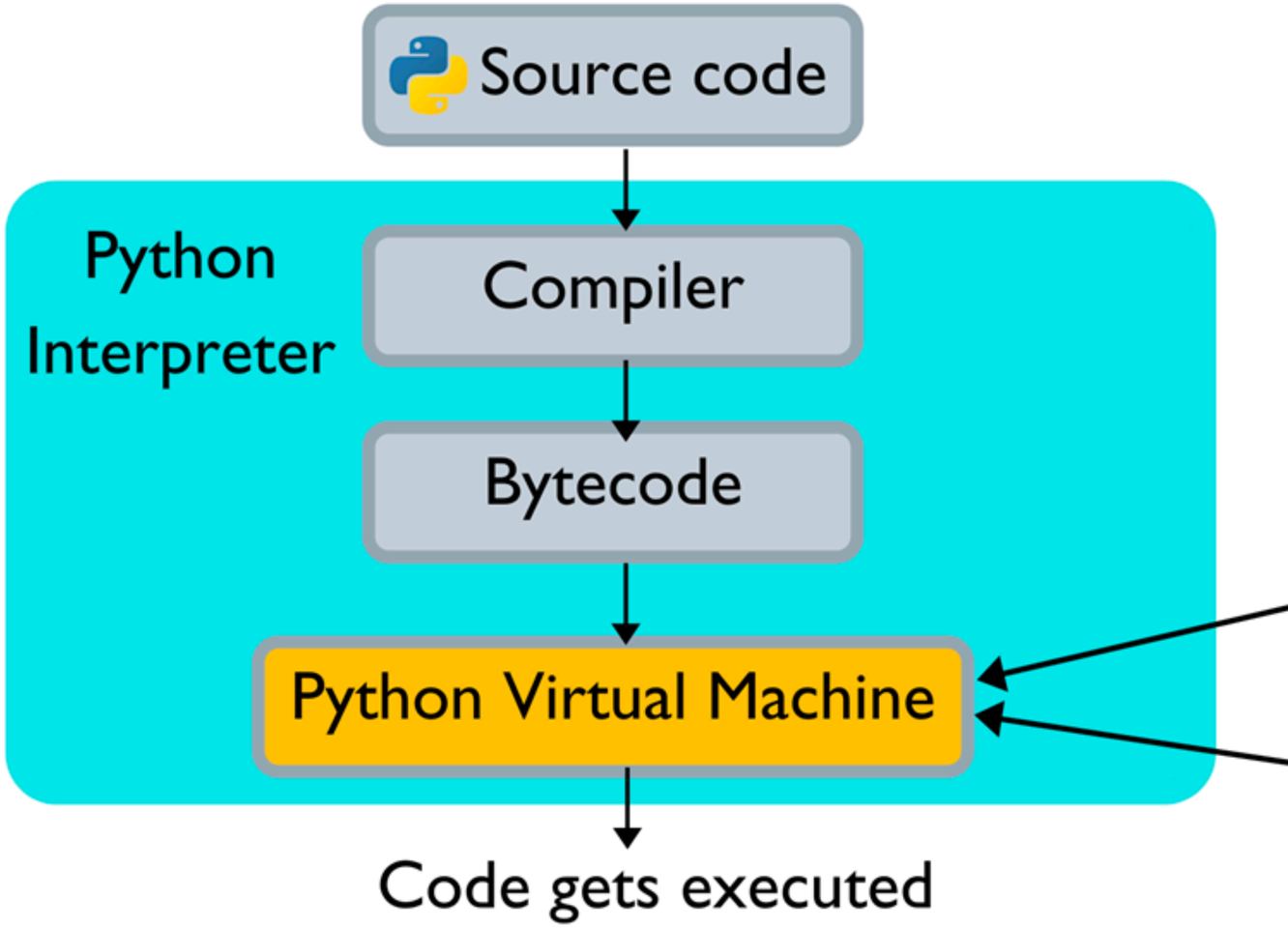
# Pros and Cons

## Interpreted

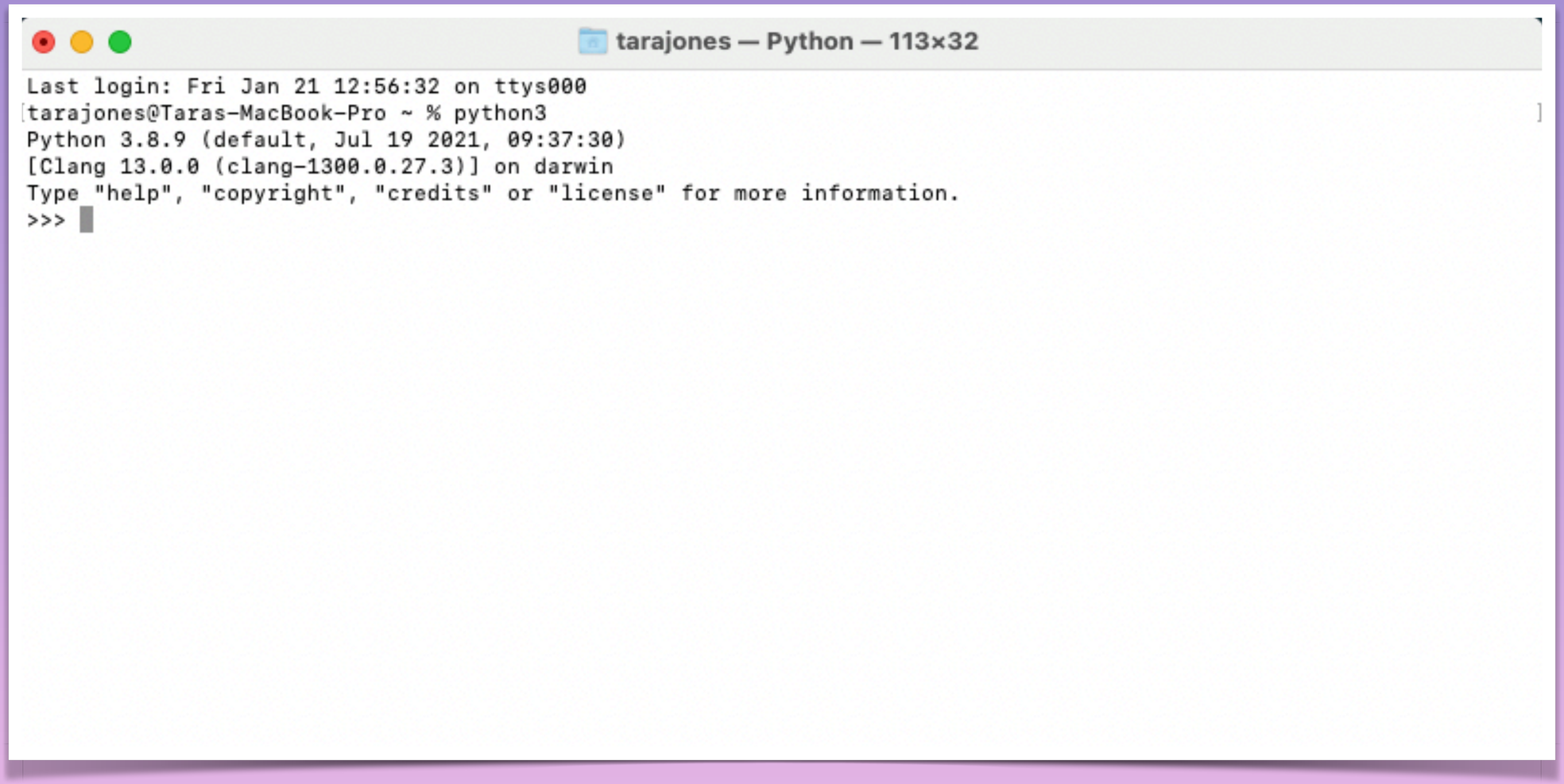
- PROS:
  - Super flexible for cross-platform (every device must interpret code on its own, never sending machine code)
  - Faster debugging, no extra step
- CONS:
  - Machines must download interpreter themselves
  - Often slower due to need to always interpret
  - Source code public

# Python Lives in the Intersection

Upfront, compile the language as close to machine code as possible while still being flexible.



Compiled: C, C++, Objective C
Interpreted: PHP, JavaScript
Hybrid: Java, C#, Python




Python being interpreted give us this great tool

# Variables in Python

# Something to get used to is no explicit distinction of types by the user

```
a = 5
name = “CS 41”
```

types are inferred by the interpreter

Now this does not mean that there are no types in Python, they are just hidden from the user and show up functionally

- Variable type can be checked by checking `type(var)`

Variables and their types can change throughout the program

# Numbers and Bools

# Numbers

```
#int
a = 5

#float
b = 5.0
```

| Numeric Operation | Operation Syntax | Assignment Syntax |
|---|---|---|
| Addition | `x + 5` | `x += 5` |
| Subtraction | `x - 5` | `x -= 5` |
| Multiplication | `x * 5` | `x *= 5` |
| Division | `x / 5` | `x /= 5` |
| Integer Division | `x // 5` | `x //= 5` |
| Modulo Operator | `x % 5` | `x %= 5` |
| Exponentiation | `x ** 5` | `x **= 5` |

# Booleans

```
cs_41 = True

is_it_friday_yet = False
```

| Boolean Operation | Operation Syntax |
|---|---|
| Not | `not a` |
| And | `a and b` |
| Or | `a or b` |
| Equals (Not Equals) | `a == b (a != b)` |
| Greater (Or Equal) | `a > b (a >= b)` |
| Less (Or Equal) | `a < b (a <= b)` |
| Chained Expressions | `a > b > c` |

# Checking value of variables

- == will check if the value of two variables are the same
- “Is” will check if the two variables are in the same memory location, or refer to the same object

# Strings

```
name = "Tara Jones"
```

```
name = "Tara Jones"
        0123456789
```

- Can parse a character by writing name of string, hard brackets, and the index:

```
name[1]
```

- Can parse a substring by writing hard brackets, and the start of the substring and the exclusive end of the substring separated by a colon:

```
name[0:4]
```

```
name = "Tara Jones"
        0123456789
```

- Can do some fun stuff with negative parsing

```
name[-1]
```

- This also works with sub string parsing

```
name[-4:-1]
```

```
name = "Tara Jones"
        0123456789
```

★ IMPORTANT to note that strings in python are immutable

- This means that they do not support changing parts of the string once it has been created
- But, we can fully reset a variable to a different string and add strings together
- Use len() to find length

# Lists

- Allow for multiple types inside
- Very similar functionality to strings
- Can check if something is “in” list

```
my_list = []
my_list.append(“CS41”)
my_list.remove(“CS41”)
my_list.append(6)
```

# Input and Casting

```
name = input(“What is your name: “)
```

```
phone_num = int(input(“What is your #?: “)
```

Questions?

# Control Flow

# if, else

```
num = int(input("Enter a number: "))
if num % 2 == 0:
  print("even")
else:
  print("odd")
```

# if, elif, else

```
animal = input("Enter an animal: ")
if animal == "dog":
  print("I love dogs!!")
elif animal == "horse":
  print("Neigh!!")
else:
  print("Cool choice!!")
```

# Truthiness and Falseness

Python objects have a quality of being “truthy” or “falsy”

- **Falsy:**
  - Variables with values of 0 or None, empty data structures
- **Truthy:**
  - Just the opposite! Variables with values other than 0 or None, data structures and strings that are not empty

# Try Except

```
name = input("What's your name?: ")

try: #dangerous code
  ID = name + 2022
except TypeError: #give an exception for a type error
  print("Oops! You tried to add two different types")
```

- For trying dangerous code

# Loops

```
for i in range(100):
  print(i)
```

```
my_list = [1,2,3,4]
for i in range(len(my_list)):
  print(my_list[i])
```

```
for elem in my_list:
  print(elem)
```

# Continue

```
#make every number odd
my_list = [1,2,3,4,5]

for elem in my_list:
  if elem % 2 == 0:
    elem+=1
  else:
    continue
```

# While Loops

```
while true:
  number = int(input("Give a number: "))
  if number > 100:
    break
```

# Functions

```
def my_func(a,b):
  return a + b
```

```
def my_func():
    print("I can take in no params and not explicitly return")
```

- Parameters do not have set type
- Return values do not have to be speficifed
- Nested functions can exist
- Return statement is option and will naturally return an object `None`

# Function Demo

Thank you!